Estágio Científico e Tecnológico I - EE015

Relatório

Predição de Séries Temporais Baseada em Redes Neurais Artificiais

Submetido à
Faculdade de Engenharia Elétrica e Computação (FEEC)

Departamento de Engenharia de Computação e Automação Industrial (DCA)
Faculdade de Engenharia Elétrica e de Computação (FEEC)
Universidade Estadual de Campinas (UNICAMP)
CEP 13083-852, Campinas - SP

Aluno: João Pedro de Oliveira Pagnan
Orientador: Prof. Levy Boccato

Campinas, 30 de junho de 2021

1 Introdução

A predição de séries temporais é uma das aplicações mais interessantes do tratamento de informação. O desafio de antecipar padrões de comportamento e construir modelos que sejam apropriados para explicar determinados fenômenos da natureza tem importância para a biologia, economia, automação industrial, meteorologia e diversas outras áreas da ciência [1].

É possível definir uma série temporal como sendo um conjunto de medidas feitas ao decorrer de um intervalo de tempo, podendo este ser contínuo ou discreto, sobre um fenômeno de interesse. Os sistemas cujas medições formam uma série temporal podem ser originados por processos determinísticos ou estocásticos [1].

Através da análise e interpretação de uma série temporal, podemos estimar os seus valores futuros, aumentando a informação que podemos obter das observações que já foram realizadas em um sistema.

Na literatura, encontramos diversos tipos de modelos para a predição de séries temporais, desde métodos clássicos lineares, como o modelo autorregressivo (AR) [1] até métodos não-lineares utilizando, por exemplo, redes neurais artificiais, sendo que dessas se destacam as redes do tipo *Multilayer Perceptron* (MLP) e as redes recorrentes, especialmente a *Long Short-Term Memory* (LSTM) [2] e a *Echo State Network* (ESN) [3].

Uma classe de sistemas dinâmicos particularmente relevante dentro do contexto de modelagem e predição de séries temporais está ligada à ideia de dinâmica caótica. Diversos fenômenos naturais, como a dinâmica populacional de uma espécie, a dinâmica atmosférica de uma região, ou até mesmo as órbitas de um sistema com três ou mais corpos celestes podem exibir comportamento caótico. Apesar de serem determinísticos (e, portanto, previsíveis), esses sistemas são extremamente sensíveis às condições iniciais [4]. Isso causa um problema para a predição das séries temporais originadas por eles, pois uma pequena incerteza na medida afetará toda a previsão.

Tendo em vista o desempenho de modelos não-lineares para previsão de diversas séries temporais [2], optamos por estudar a aplicabilidade de redes neurais artificiais à previsão de séries relacionadas a sistemas com dinâmica caótica.

Essa primeira parte do projeto de iniciação científica teve como objetivo estudar a base teórica das redes neurais artificiais e de outros regressores lineares clássicos, assim como estudar os fundamentos de sistemas dinâmicos e de dinâmica caótica. Os principais modelos estudados, juntamente com uma breve exposição de modelos lineares básicos, são apresentados na Seção 2. Já na Seção 3, veremos alguns conceitos fundamentais e a caracterização de sistemas caóticos.

O estudo dirigido começou abordando uma revisão de tópicos de probabilidade, teoria da informação e estimação. Em seguida, foi vista a teoria de regressores e classificadores clássicos [5]. Depois disso, o estudo se dirigiu para as redes neurais artificiais MLP e recorrentes [6], para, por fim, concluir o aprendizado de preditores com uma breve exposição dos modelos autorregressivos (AR) e autorregressivos de médias móveis (ARMA) [1]. Com a teoria de predição solidificada, o foco mudou para

os fundamentos da teoria de sistemas com dinâmica caótica, utilizando como base as referências [4] e [7].

Por fim, na Seção 4 são indicados os próximos passos deste projeto de iniciação científica visando sua conclusão ao final deste primeiro semestre de 2021.

2 Modelos de Predição

2.1 Modelos Lineares

Apesar desta pesquisa focar na aplicabilidade de modelos preditores não-lineares utilizando redes neurais artificiais, é pertinente darmos uma introdução ao assunto através de modelos lineares para essa aplicação, já aproveitando o momento para apresentarmos alguns conceitos básicos de predição.

2.1.1 Modelo Autorregressivo (AR)

No modelo autorregressivo (AR, do inglês *autoregressive*) o valor da série para um instante de tempo n, denotado por $x(n)$, é dado pela combinação linear dos valores passados a partir do instante $n - L - (K - 1)$ até o instante $n - L$, onde L é o passo de predição (quantos instantes de tempo à frente pretende-se predizer o valor da série) e K é a ordem do modelo.

Portanto, podemos dizer que, no modelo AR o valor da série temporal num instante n é dado por [8]:

$$x(n) = a_1 \cdot x(n - L) + a_2 \cdot x(n - L - 1) + ... + a_K \cdot x(n - L - (K - 1)) + \eta(n) \qquad (1)$$

onde a_k, $k = 1, 2, ..., K$ são os coeficientes que ponderam as amostras nos instantes passados e $\eta(n)$ é o erro instantâneo do modelo preditor. Esse erro instantâneo é um ruído branco (do inglês, *white noise*), possuindo média nula e variância σ_n^2 constante [1].

É interessante mencionar que, se considerarmos $L = 1$, ou seja, se estivermos predizendo o valor da série num instante seguinte ao atual, podemos dizer que:

$$\sum_{k=0}^{K} w_k \cdot x(n - k) = \eta(n) \qquad (2)$$

sendo $w_0 = 1$ e $w_k = -a_k$ para $1 \leq k \leq K$.

Perceba que o lado esquerdo de (2) é uma soma de convolução em tempo discreto, portanto, podemos interpretar o modelo AR como um sistema linear e invariante com o tempo (LIT) [8].

2.1.2 Modelo Autorregressivo e de Médias Móveis (ARMA)

O modelo ARMA (do inglês, *auto-regressive moving-average*) também leva em consideração o valor do ruído branco nos instantes de tempo anteriores ao atual [1]:

$$x(n) = \sum_{k=1}^{K} a_k \cdot x(n - L - (k-1)) + \sum_{k=1}^{M} b_k \cdot \eta(n - L - (k-1)) \tag{3}$$

Para ser parametrizado, o modelo ARMA necessita de métodos iterativos e/ou heurísticos. Isso é devido ao fato de que não há soluções em forma fechada para obter os coeficientes b_k. Além disso, é válido mencionar que durante a otimização desses parâmetros, devemos nos atentar a estabilidade desse sistema, afinal, os erros podem se acumularem, levando a uma divergência na saída do preditor [1].

Apesar dos modelos lineares terem e ainda serem bastante utilizados para a predição de séries temporais, para determinadas situações a regra linear aplicada pelos modelos AR e ARMA não é suficiente para realizar uma predição com um erro aceitável em sistemas mais complexos.

Devido a isso, optamos por direcionar a análise para modelos não-lineares, nesse caso, utilizando redes neurais artificiais para a predição. Veremos então como são esses tipos de preditores.

2.2 Modelos Não-lineares

Os modelos não-lineares estudados foram as famosas redes neurais artificiais.

As redes neurais artificiais são ferramentas computacionais cujas estruturas são inspiradas no funcionamento das redes neurais biológicas presentes em cérebros de animais desenvolvidos, em especial do ser humano. Podemos interpretar um neurônio (tanto biológico, quanto artificial) como uma unidade de processamento de informação [9].

Analogamente, uma rede neural artificial é uma estrutura formada por vários neurônios artificiais interconectados, a qual é capaz de processar estímulos (sinais) de entrada e de produzir respostas conforme a tarefa desejada. Existem alguns modelos matemáticos para o neurônio artificial, sendo o *perceptron* um dos mais usuais (vide Seção 2.2.1). Além disso, os neurônios podem ser organizados de diferentes maneiras para construir a arquitetura (ou topologia) da rede neural, a qual é tipicamente estruturada em camadas. Por fim, os neurônios artificiais podem exibir uma estrutura interna que varia de acordo com a arquitetura desejada para a aplicação, como observaremos na Seção 2.2.2, onde são discutidos alguns exemplos de redes recorrentes.

2.2.1 Redes *Multilayer Perceptron* (MLP)

Um dos modelos mais utilizados para representar um neurônio artificial, o *Perceptron* [10], é apresentado na Figura 1.

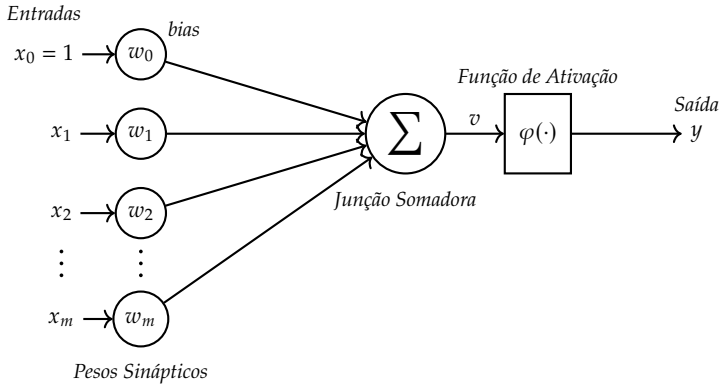

Figura 1: Modelo *Perceptron* para o neurônio artificial

Em termos matemáticos, a saída do neurônio pode ser escrita como:

$$y = \varphi(v) = \varphi\Big(\sum_{i=1}^{m} w_i x_i + w_0\Big) = \varphi\Big(\sum_{i=0}^{m} w_i x_i\Big) = \varphi(\mathbf{w}^T \cdot \mathbf{x}), \tag{4}$$

onde $\mathbf{w}$ é o vetor que contém os coeficientes, denominados de pesos sinápticos, que ponderam as entradas do neurônio.

A escolha da função de ativação $\varphi(\cdot)$ varia de acordo com a aplicação desejada. Ela pode ser desde uma função de *Heaviside*, a puramente linear $\varphi(x) = x$, ou até mesmo a tangente hiperbólica, a função logística ou outras funções não-lineares para mapeamentos mais complexos [6]. Na figura 2, vemos alguns exemplos de funções de ativação comumente utilizadas nos neurônios *Perceptron*, assim como as suas derivadas.

É interessante mencionar o fato de que, assim como podemos selecionar uma gama de funções não-lineares para a ativação do neurônio, de forma a realizar transformações não-lineares na entrada, também é possível utilizar funções de ativação lineares, ou seja, realizar transformações lineares assim como os modelos clássicos de predição (como o AR e o ARMA) fazem. Nesse caso, a falta da não-linearidade tornaria muito difícil ou até mesmo impossibilitaria que sistemas mais complexos fossem descritos de forma aceitável por redes neurais artificiais desse tipo [11].

Tipicamente, uma rede neural MLP é composta por um número arbitrário N_L de camadas com n neurônios do tipo *Perceptron*, com a característica de que as saídas dos neurônios da l-ésima camada são propagadas para a frente, servindo como as entradas de todos os neurônios da camada seguinte $(l + 1)$. Esse processo é conhecido como *feedforward*. Por isso, este tipo de rede é conhecida como totalmente conectada (ou densa). A figura 3 apresenta a estrutura típica das redes MLP.

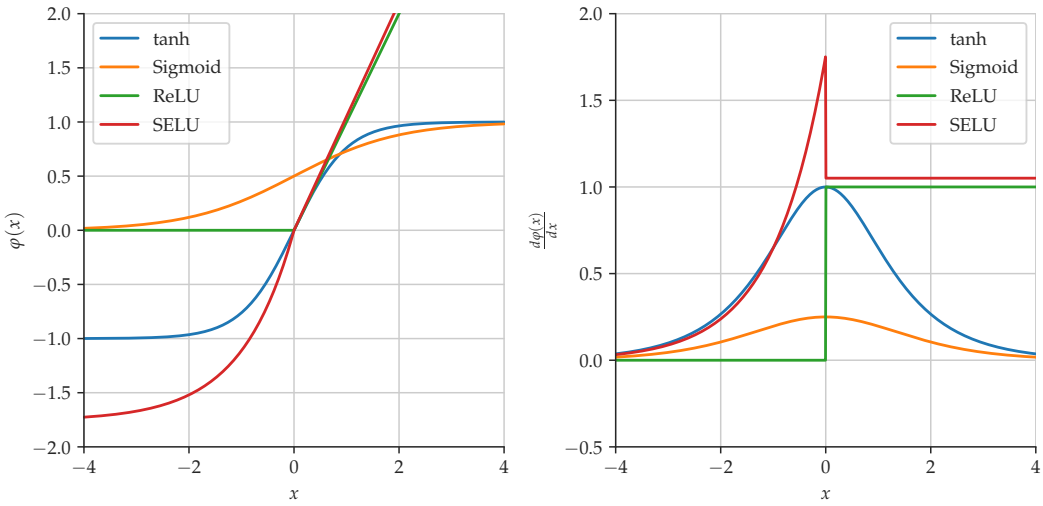

Figura 2: À esquerda, algumas funções de ativação comuns em tarefas de predição e regressão para o neurônio *Perceptron* e, à direita, suas derivadas

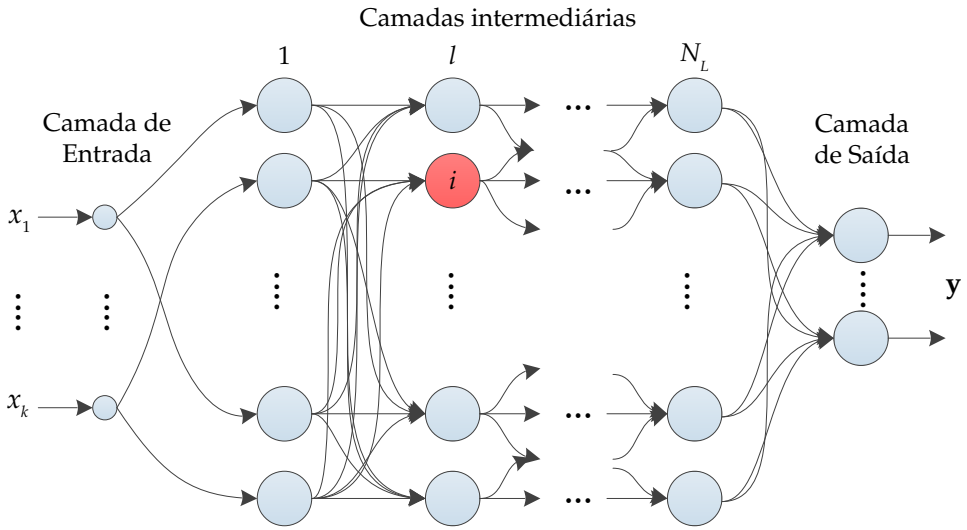

Figura 3: Estrutura típica de uma rede MLP (figura adaptada de [12])

Pela figura vemos que, além das N_L camadas intermediárias com neurônios *Perceptron*, a estrutura das redes MLP contém uma camada de entrada, que possui ativação linear e apenas passa o atributo de entrada relacionado ao neurônio em específico (x_1, x_2, ..., x_k) para a primeira camada intermediária, e uma camada de saída que gera as respostas da rede neural para um vetor de entrada. A função de ativação da camada de saída e o número de neurônios presente nela dependem da tarefa a ser realizada pela MLP. Por exemplo, em aplicações de regressão é comum o uso de um ou mais neurônios (dependendo se a saída do preditor será um vetor com um ou mais instantes de tempo) com função de ativação linear. Já em aplicações de classificação, a função de ativação pode variar entre a função logística (utilizada no caso binário em que queremos saber se uma entrada pertence ou não a uma classe) e entre a função *softmax* (que gera uma saída para cada classe, indicando a probabilidade de uma entrada pertencer à classe em específico) e, novamente, o número de neurônios de saída também varia com o tipo da classificação.

De forma similar a feita em (4), é possível representar a saída do i-ésimo neurônio da l-ésima camada intermediária, sendo que a anterior a esta possui n_{l-1} neurônios, a l-ésima possui n_l neurônios e a estrutura possui N_L camadas intermediárias, da seguinte forma:

$$y_i^l = \varphi^l\Big(\sum_{j=1}^{n_{l-1}} w_{ij}^l y_j^{l-1} + w_{i0}^l\Big) \tag{5}$$

onde w_{ij}^l representa o peso sináptico da conexão que liga o j-ésimo neurônio da camada $l-1$ ao i-ésimo neurônio da camada l, sendo que na primeira camada intermediária os sinais de entrada são os atributos do vetor de entrada, ou seja, $y_j^0 = x_j$ com $j = 1, ..., k$ [12].

O processo de treinamento de uma rede neural artificial normalmente é realizado com sequências de vetores de entrada $\mathbf{x}$, chamadas de *mini-batch*, e chamamos um período de treinamento de época (do inglês *epoch*) [6]. No treinamento, os pesos sinápticos $\mathbf{w}$ são ajustados em um processo iterativo de forma a minimizar uma função custo $J(\mathbf{w})$ que representa uma medida do erro entre as saídas geradas pela rede e as saídas desejadas (vale mencionar que $\mathbf{w}$ é um vetor com todos os parâmetros da rede). No caso de um problema de regressão ou predição, é comum que a função custo a ser minimizada seja o Erro Quadrático Médio (MSE, do inglês *Mean Squared Error*), assim, o problema envolve otimização não-linear irrestrita [9].

Para isso, é frequente o uso de algoritmos de otimização baseados em derivadas da função custo $J(\mathbf{w})$, como o método do gradiente descendente estocástico (SGD, do inglês *stochastic gradient descent*), o método de Nesterov (NAG, do inglês *Nesterov Accelerated Gradient*) e o algoritmo Adam (*Adaptive Moment Estimation*) [6]. O famoso algoritmo de retropropagação (*backpropagation*) do erro, cuja representação matemática também varia de acordo com o tipo de otimização, é empregado para viabilizar o cálculo das derivadas com relação aos pesos sinápticos dos neurônios situados nas camadas internas da rede, de forma que, para cada *mini-batch* sejam atualizados todos os pesos de todos os neurônios presentes em todas as camadas.

Nesse caso, é comum dividirmos os métodos de otimização entre métodos de primeira ordem e de segunda ordem. Os métodos de primeira ordem utilizam as derivadas de primeira ordem da função custo, geralmente representadas com o uso do vetor gradiente:

$$\nabla J(\mathbf{w}) = \Big[\frac{\partial J(\mathbf{w})}{\partial w_1} \ \ldots \ \frac{\partial J(\mathbf{w})}{\partial w_n}\Big]^T \tag{6}$$

assim, ao caminharmos na direção contrária à apontada pelo vetor $\nabla J(\mathbf{w})$ obtemos, de forma iterativa, a minimização desejada. Logo, a regra de atualização dos pesos pode ser dada pela forma básica:

$$\mathbf{w}[k+1] \leftarrow \mathbf{w}[k] - \eta \nabla J\big[\mathbf{w}[k]\big] \tag{7}$$

Os algoritmos de otimização mencionados (SGD, Adam, NAG) utilizam variações da regra de atualização de $\mathbf{w}$ indicada em (7).

Para esta pesquisa, optamos por utilizar o algoritmo Nadam (*Nesterov Adaptive*

Moment Estimation) [13]. O Nadam, como o nome já sugere, incorpora elementos dos algoritmos NAG e Adam, de forma a ter um passo de atualização adaptativo e com acúmulo dos gradientes mais recentes (semelhante a ideia de momento linear de uma partícula), com a utilização do "truque de Nesterov", desenvolvido pelo matemático russo Yurii Nesterov [14], onde o gradiente não é calculado sobre o ponto atual indicado pelo vetor $\mathbf{w}$, mas sim sobre um ponto levemente à frente, na mesma direção do momento. O conjunto de equações que descreve o algoritmo Nadam é dado por:

$$\mathbf{w}' = \mathbf{w} + \beta_3 \mathbf{m} \tag{8a}$$

$$\mathbf{m} \leftarrow \beta_1 \mathbf{m} - (1 - \beta_1) \nabla_{\mathbf{w}} E(\mathbf{w}') \tag{8b}$$

$$\mathbf{s} \leftarrow \beta_2 \mathbf{s} - (1 - \beta_2) \nabla_{\mathbf{w}} E(\mathbf{w}') \otimes \nabla_{\mathbf{w}} E(\mathbf{w}') \tag{8c}$$

$$\hat{\mathbf{m}} \leftarrow \frac{\mathbf{m}}{1 - \beta_1} \tag{8d}$$

$$\hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1 - \beta_2} \tag{8e}$$

$$\mathbf{w} \leftarrow \mathbf{w} + \eta \hat{\mathbf{m}} \oslash \sqrt{\hat{\mathbf{s}} + \epsilon} \tag{8f}$$

A incorporação dos gradiantes passados no cálculo do próximo vetor $\mathbf{w}$ é através da equação (8b), que define o vetor de momento $\mathbf{m}$. Assim, utilizando a analogia com o momento linear de uma partícula, o gradiente passa ter a ideia de uma força de aceleração que aumenta o valor do momento do vetor $\mathbf{w}$ caso este esteja indo para um ponto ótimo de mínimo local, escapando mais rapidamenete de regiões onde não há minimização significativa da função custo.

O hiperparâmetro β_1 presente em (8b) serve tanto para evitar que $\mathbf{m}$ aumente de forma descontrolada, como também para fazer com que, através do termo $(1 - \beta_1)$, uma média exponencialmente decrescente de valores anteriores. Normalmente, β_1 é inicializado em 0.9.

Já o vetor $\mathbf{s}$, definido em (8c), serve para evitar que o algoritmo de otimização caminhe com uma direção que não aponte para o ponto ótimo mencionado anteriormente. Além disso, para evitar que esse ajuste na direção cause uma perda significativa na rapidez de convergência do algoritmo, utiliza-se os termos β_2 e $(1 - \beta_2)$, ou seja, um decaimento exponencial nos valores passados de $\mathbf{s}$. Nesse caso, o hiperparâmetro β_2 é inicializado em 0.999 e ϵ em 10^{-7} (para evitar uma divisão por zero).

Por fim, as equações (8d) e (8e) servem para inicializar os vetores $\mathbf{m}$ e $\mathbf{s}$ com valores próximos de 1, impulsionando a atualização deles no começo do treinamento.

É importante mencionar que os métodos de otimização aqui mencionados são métodos de busca local, ou seja, têm convergência esperada para um mínimo local, que não necessariamente é o mínimo global da função custo para a aplicação. Dizemos também que esses pontos mínimos possuem "bacias de atração"que, para valores adequados da taxa de aprendizado η, atrai o vetor de parâmetros $\mathbf{w}$ [5].

Algo presente durante o treinamento da rede neural é, ou desaparecimento dos

gradientes (do inglês *vanishing gradients problem*) nas camadas inferiores da rede, ou o aumento desenfreado desses gradientes (do inglês *exploding gradients problem*) nas camadas citadas, que é mais comum para redes recorrentes. Para evitar que isso ocorra, normalmente são utilizadas camadas que realizam uma normalização do *batch* de vetores de entrada (BN, do inglês *Batch Normalization*), sendo posicionadas antes ou depois das camadas intermediárias da rede [6].

O que a normalização faz, é centrar todas as instâncias de vetores de entrada presentes no atual *mini-batch* ao redor do zero e as normaliza. Logo, podemos representar a transformação aplicada com as seguintes equações:

$$\boldsymbol{\mu}_B = \frac{1}{d}\sum_{i=1}^{d}\mathbf{x}_i \tag{9a}$$

$$\sigma_B^2 = \frac{1}{d}\sum_{i=1}^{d}(\mathbf{x}_i - \boldsymbol{\mu}_B)^2 \tag{9b}$$

$$\hat{\mathbf{x}}_i = \frac{\mathbf{x}_i - \boldsymbol{\mu}_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{9c}$$

$$\mathbf{z}_i = \boldsymbol{\gamma} \otimes \hat{\mathbf{x}}_i + \boldsymbol{\beta} \tag{9d}$$

sendo $\boldsymbol{\mu}_B$ e σ_B os vetores com as médias e desvio-padrão para cada atributo de entrada do atual *mini-batch* (este contendo d vetores $\mathbf{x}$), $\hat{\mathbf{x}}_i$ é o i-ésimo vetor de entrada centrado em zero e normalizado, $\boldsymbol{\gamma}$ é o vetor que pondera cada um dos atributos de $\hat{\mathbf{x}}_i$, $\boldsymbol{\beta}$ tem um termo de *offset* para cada atributo normalizado, ϵ novamente é um termo pequeno (geralmente na ordem de 10^{-5}) para evitar uma divisão por zero e, por fim, $\mathbf{z}_i$ é a saída da camada de *batch normalization*.

Os hiperparâmetros $\boldsymbol{\beta}$ e $\boldsymbol{\gamma}$ controlam o deslocamento e o escalonamento, respectivamente, de cada atributo para um vetor de entrada. Essas variáveis são parametrizadas durante o treinamento da rede neural de forma bem eficiente dependendo da quantidade e do tamanho dos *mini-batches*.

Também vale mencionar que, para o conjunto de dados de validação, ou seja, o conjunto de dados que, após o ajuste dos pesos $\mathbf{w}$ no conjunto de treinamento, é verificado se a rede neural está generalizando os resultados, evitando o sobreajuste (do inglês *overfitting*), são calculados outros vetores de valores de média ($\boldsymbol{\mu}'_B$) e desvio-padrão (σ'_B). Assim, o processo descrito anteriormente também é realizado sobre esses dados.

Por fim, o grande apelo das redes *Multilayer Perceptron* é que elas tem a capacidade de aproximação universal, ou seja, são capazes de aproximar qualquer mapeamento contínuo num domínio compacto com um nível de erro arbitrariamente pequeno. Até mesmo uma MLP com uma única camada intermediária e camada de saída linear já possui esta capacidade [15, 11]. Infelizmente, esse teorema não indica a quantidade de neurônios necessária na(s) camada(s) intermediária(s), muito menos um método para ajustar o vetor $\mathbf{w}$ da rede para garantir a solução ótima.

Além disso, para o propósito deste trabalho de pesquisa, que envolve sistemas dinâmicos, ter acesso a um histórico e a uma memorização dos padrões anteriores de entrada, na teoria, aumentaria a precisão da predição das séries temporais. Mesmo que as redes MLP tenham capacidade de aproximação universal, é possível aumentar a eficiência desse processo caso fossem utilizadas estruturas recorrentes [2, 12].

Logo, incluímos na análise alguns modelos de redes neurais recorrentes. Discutiremos mais sobre a estrutura e o funcionamento delas na próxima seção.

2.2.2 Redes Neurais Recorrentes (RNN)

Diferentemente das redes MLP que são *feedforward*, ou seja, que não reutilizam a informação processada dos padrões anteriores para gerar a próxima saída, a ideia central das redes neurais recorrentes (RNN, do inglês *Recurrent Neural Networks*) é que elas têm estruturas computacionais que podem armazenar os estados anteriores dos neurônios, podendo também possuir portas não-lineares que regulam o fluxo de informação de entrada e de saída da célula computacional [9]. Uma representação possível de uma célula básica de uma rede recorrente pode ser vista na Figura 4 (esquerda).

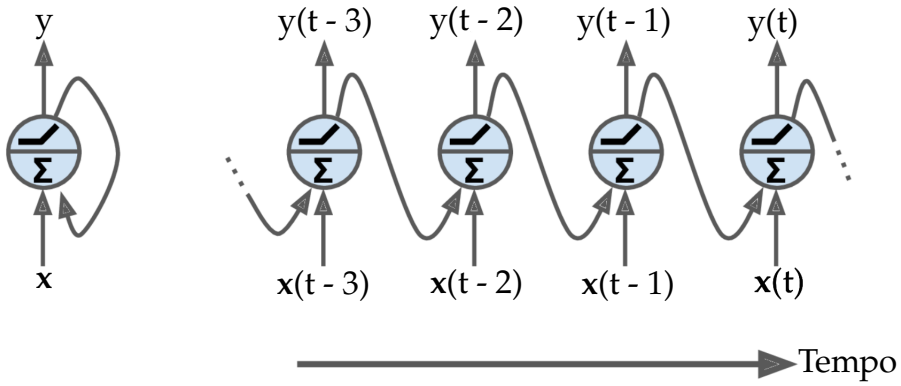

Figura 4: À esquerda, célula básica da rede recorrente e, à direita, sua representação através do tempo (figura adaptada de [6])

Note que a saída é realimentada (com um atraso temporal) para a entrada do próprio neurônio, assim a célula recorrente recebe tanto o vetor de entradas $\mathbf{x}(t)$ assim como a saída no instante anterior $y(t-1)$ que, devido a relação de recorrência, traz as saídas em todos os instantes anteriores $y(t-k)$ com $1 < k \leq t$ (considerando $y(0)$). Dessa forma, como pode ser visto na Figura 4 (direita), é possível representar essa rede através do tempo, através de um processo chamado desenrolamento da rede no tempo (do inglês *unrolling the network through time*) [6].

Uma camada recorrente é formada de maneira similar à célula, com a diferença que cada neurônio recebe, além do vetor de entradas $\mathbf{x}(t)$, um vetor de saídas $\mathbf{y}(t-1)$, que contém as saídas de todos as unidades recorrentes da camada em instantes anteriores, conforme a Figura 5 indica.

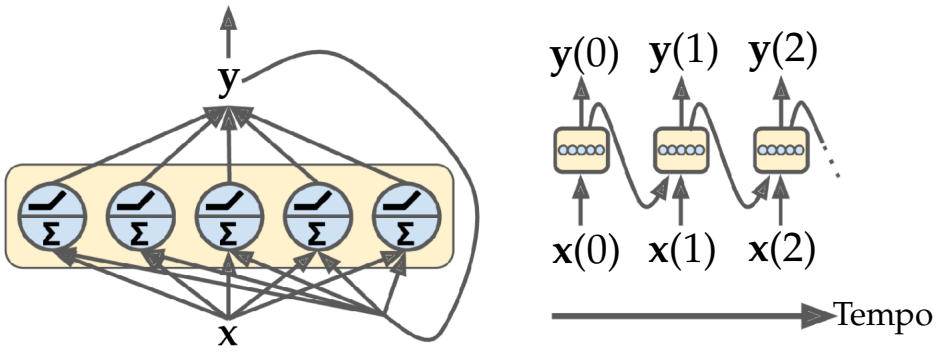

Figura 5: À esquerda, camada da rede recorrente e, à direita, sua representação através do tempo (figura adaptada de [6])

Dessa forma, cada célula recorrente possui dois vetores de pesos, $\mathbf{w}_x$ e $\mathbf{w}_y$. O primeiro pondera o vetor de entradas $\mathbf{x}(t)$, de forma similar à apresentada anteriormente na MLP. Já o vetor $\mathbf{w}_y$ pondera o vetor das saídas dos estados passados $\mathbf{y}(t-1)$.

Logo, pode-se formar uma camada recorrente agrupando os vetores de pesos de cada célula nas matrizes $\mathbf{W}_x$ e $\mathbf{W}_y$, sendo que a saída da camada, isto é, $\mathbf{y}(t)$, é dada por:

$$\mathbf{y}(t) = \varphi\big(\mathbf{W}_x\mathbf{x}(t) + \mathbf{W}_y\mathbf{y}(t-1) + \mathbf{b}\big) \tag{10}$$

Assim, como o vetor $\mathbf{y}(t-1)$ presente em (10) contém, implicitamente, a saída da camada no instante t acaba sendo influenciada por todas as entradas anteriores ($\mathbf{x}(0)$, $\mathbf{x}(1)$, ... $\mathbf{x}(t-1)$). Devido a isso, é dito que a rede recorrente possui uma memória dos estados anteriores [6].

Em geral, o estado atual de uma célula é representado através de uma função $\mathbf{h}(t)$, de forma a termos a seguinte relação entre $\mathbf{h}(t)$ e $\mathbf{y}(t)$:

$$\mathbf{h}(t) = f(\mathbf{x}(t), \mathbf{h}(t-1)) \tag{11}$$

$$\mathbf{y}(t) = g(\mathbf{x}(t), \mathbf{h}(t-1)) \tag{12}$$

Nas células mais básicas apresentadas anteriormente, temos que $f(\mathbf{x}(t), \mathbf{h}(t-1)) = g(\mathbf{x}(t), \mathbf{h}(t-1))$, mas esse nem sempre é o caso. Assim, uma representação mais geral de uma célula recorrente pode ser vista na Figura 6.

A saída de uma RNN varia com a aplicação. Para a predição de séries temporais, utilizamos o arranjo sequência para sequência (do inglês *sequence-to-sequence network*), que a rede recebe uma sequência de entradas e gera uma sequência de saídas. Por exemplo, sendo $x(t)$ o valor da série temporal, $y(t)$ é a estimação da rede para o instante $t+L$, sendo L o passo de predição. Logo, fornecendo em $t=0$ vetores de entrada um a um, as saídas serão as estimações dos valores para os instantes 1 a $N+L$, sendo N o número de amostras da série.

Um outro tipo de saída possível para a predição de séries temporais é o arranjo sequência para vetor (do inglês *sequence-to-vector network*), onde ignoramos todas as

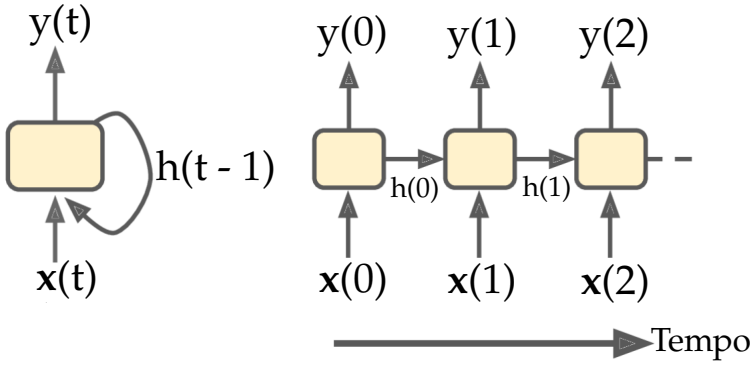

Figura 6: À esquerda, representação geral da célula da rede recorrente e, à direita, sua representação através do tempo (figura adaptada de [6])

saídas com exceção da última. Assim, com esse arranjo é possível predizer os próximos L valores desconhecidos da série. Essa configuração também é muito utilizada para gerar, com base em uma frase, um *score* para o sentimento que foi transmitido pelo autor [6].

Além disso, também é possível termos um arranjo vetor para sequência (do inglês *vector-to-sequence network*). Esse tipo não é tanto utilizada para a predição de séries temporais, sendo mais utilizada para gerar legendas para imagens.

Por fim, podemos combinar o esquema sequência-vetor seguido do vetor-sequência. Essa configuração é chamada de *Encoder-Decoder*, utilizado para gerar traduções com base em uma frase. O *Encoder* recebe uma sequência de palavras como entrada, gerando um vetor que é decodificado pelo *Decoder* para formar o texto em outro idioma. A Figura 7 exibe as configurações possíveis.

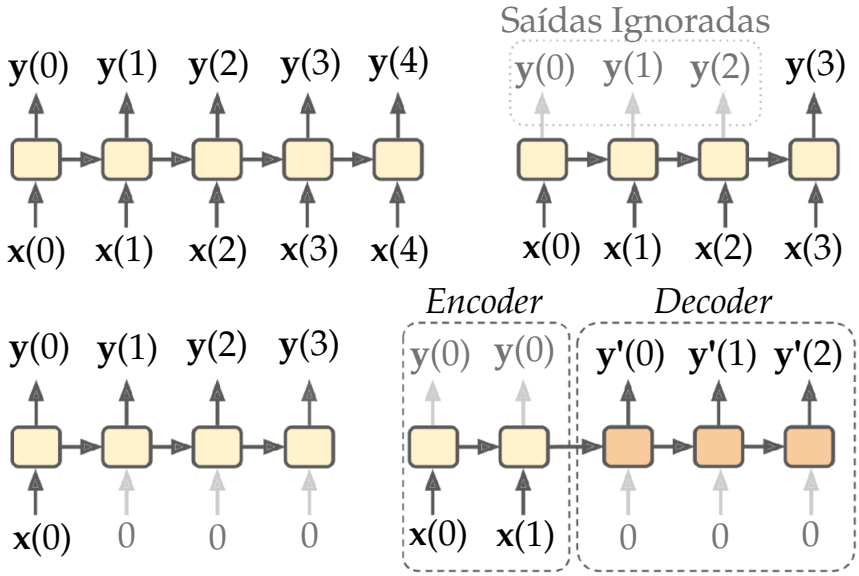

Figura 7: Maneiras de se implementar uma RNN (figura adaptada de [6])

O treinamento de uma rede recorrente também utiliza a técnica de retropropagação do erro, citada anteriormente quando foi discutida a rede MLP. A diferença é que o

erro também é retropropagado para k instantes anteriores de tempo. Dessa forma, essa técnica é chamada de retropropagação do erro através do tempo (BPTT, do inglês *backpropagation through time*).

Para compreender os pontos principais do funcionamente desse algoritmo, convém considerar a rede desdobrada no tempo (Figura 5). Dessa forma, pode-se perceber que uma rede recorrente desdobrada no tempo é bem semelhante a uma rede *feedforward* profunda, em que as várias camadas que representam a rede em instantes de tempo diferente compartilham os mesmos parâmetros.

Nesse caso, a entrada de camada camada da rede desdobrada corresponde ao vetor de entrada para um determinado instante de tempo. Logo, a camada correspondente ao primeiro instante de tempo recebe a entrada $\mathbf{x}(t-k)$, a camada seguinte a ela recebe a entrada $\mathbf{x}(t-k+1)$, e isso ocorre até a camada mais recente no tempo da rede, que recebe a entrada $\mathbf{x}(t)$. Também é válido mencionar que as saídas de determinados instantes de tempo podem ser ignoradas ao calcular a função custo J para a rede desdobrada, assim como indica a Figura 8.

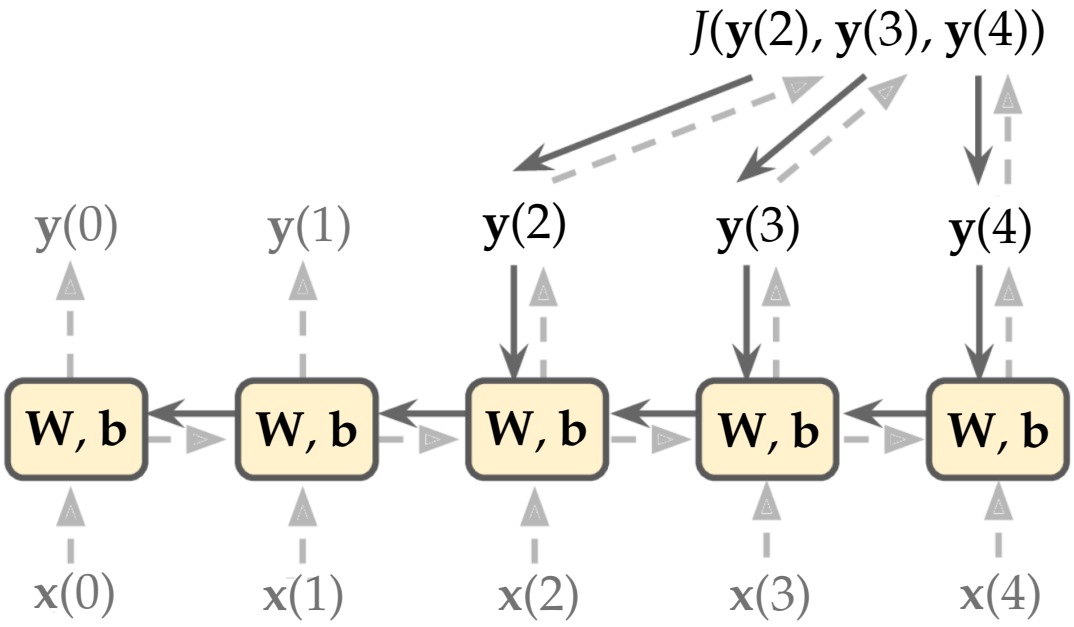

Figura 8: Retropropagação do erro através do tempo (figura adaptada de [6])

Logo, o algoritmo calcula a função custo J, considerando as saídas relevantes, e os gradientes dessa função para os pesos $\mathbf{W}$ e $\mathbf{b}$ são propagados em direção aos instantes de tempo anteriores (ignorando os instantes com saídas irrelevantes) e, como esses pesos são os mesmos em todas as camadas da rede desdobrada, eles são ajustados combinando a direção do gradiente calculado em cada instante de tempo. Essa é a essência da retropropagação do erro através do tempo.

É importante mencionar que essa técnica também está sujeita a obstáculos semelhantes aos mencionados anteriormente sobre a retropropagação do erro tradicional, como gradientes explosivos e/ou desvanecimento dos mesmos. Além disso, como

a rede recorrente é um sistema dinâmico, podem ocorrer problemas com relação à instabilidade do processo de treinamento.

Para mitigar esses problemas, são utilizadas técnicas de *gradient clipping* e normalizações [6], além do uso de funções de ativação baseadas nas funções tangente hiperbólica (tanh) e logística (Sigmoid), devido às saturações das mesmas para entradas muito grandes, ou muito pequenas (Figura 2).

A célula recorrente apresentada aqui é bem básica, aprendendo apenas padrões curtos. Assim, na seção seguinte apresentaremos a rede recorrente LSTM, que possui uma estrutura mais elaborada e é capaz de aprender padrões mais complexos.

2.2.3 Redes *Long Short-term Memory* (LSTM)

Devido às transformações que uma rede recorrente convencional aplica em uma entrada, parte da informação original transmitida é perdida e, depois de várias iterações, não sobra traços das primeiras entradas.

As redes LSTM (do inglês, *Long Short-term Memory*) contornam esse problema inserindo portas dentro da célula recorrente que controlam o fluxo de informação [16].

O neurônio da rede LSTM é bem semelhante ao neurônio básico da rede recorrente mostrado anteriormente. A diferença é a presença de um vetor de longo prazo **c** e um vetor de curto prazo **h**, conforme indicado na Figura 9.

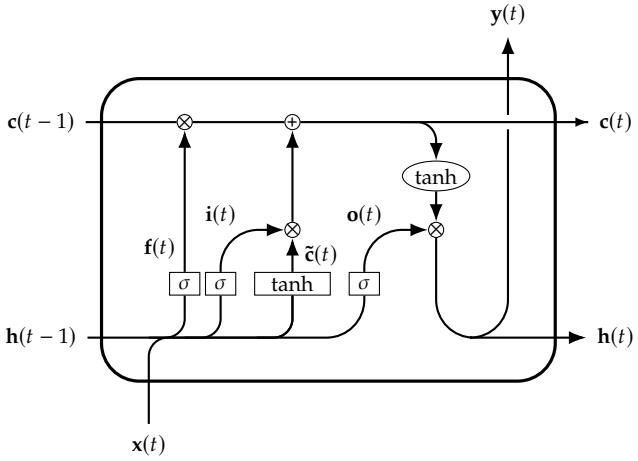

Figura 9: Estrutura interna de uma célula LSTM

Durante o processo de treinamento, a rede aprende o que deve ser armazenado no vetor $\mathbf{c}(t)$ e, do que foi guardado, o que deve ser descartado e o que deve ser levado em conta na hora de estimar a próxima saída.

A primeira operação realizada nesse vetor é a multiplicação elemento a elemento

($\otimes$) com o vetor $\mathbf{f}(t)$. Essa operação também é chamada de porta do esquecimento (do inglês, *forget gate*) pois ela determina quais elementos de $\mathbf{c}(t-1)$ serão descartados, ou seja, quais memórias de instantes passados no longo prazo serão esquecidas.

Após a incorporação de novas informações na operação de soma elemento a elemento ($\oplus$), o vetor $\mathbf{c}(t-1)$ é mandado para a saída da célula e uma cópia sua passa pela função tangente hiperbólica e é filtrado pela porta de saída (do inglês, *output gate*), formando o vetor de curto prazo $\mathbf{h}(t)$, que é também a saída $\mathbf{y}(t)$.

A entrada do neurônio LSTM, além de incluir o vetor $\mathbf{c}(t)$, também inclui os vetores $\mathbf{h}(t-1)$ e $\mathbf{x}(t)$, e com eles são formados os vetores $\mathbf{f}(t)$, $\mathbf{i}(t)$, $\tilde{\mathbf{c}}(t)$ e $\mathbf{o}(t)$.

Perceba que o vetor $\tilde{\mathbf{c}}(t)$ é a saída de uma célula recorrente básica, considerando a tangente hiperbólica como função de ativação. Ou seja, ela é composta pela ponderação entre os estados anteriores de curto prazo $\mathbf{h}(t)$ e a entrada atual $\mathbf{x}(t)$, além do vetor de *bias* $\mathbf{b}(t)$. A LSTM guarda as partes mais relevantes do vetor $\tilde{\mathbf{c}}(t)$ no vetor $\mathbf{c}(t-1)$ através da soma elemento a elemento mencionada anteriormente.

Os outros vetores restantes representam operações de controle do fluxo de informação dentro do neurônio, cada termo variando de 0 a 1 devido a função logística (σ). Assim, ao serem multiplicadas elemento a elemento, controlam quais informações serão eliminadas.

No caso, $\mathbf{f}(t)$ controla quais partes de $\mathbf{c}(t-1)$ serão apagadas, $\mathbf{i}(t)$ controla quais informações de $\tilde{\mathbf{c}}(t)$ serão agregadas ao vetor $\mathbf{c}(t-1)$ e o vetor $\mathbf{o}(t)$ controla quais componentes de $\mathbf{c}(t-1)$ deverão formar a saída $\mathbf{y}(t)$ e o vetor de curto prazo $\mathbf{h}(t)$.

O conjunto de equações abaixo mostra as operações presentes na estrutura interna da célula LSTM, assim como a saída para o instante $\mathbf{y}(t)$, presente no vetor $\mathbf{h}(t)$:

$$\mathbf{f}(t) = \sigma(\mathbf{W}_f[\mathbf{h}(t-1), \mathbf{x}(t)] + \mathbf{b}_f) \tag{13a}$$

$$\mathbf{i}(t) = \sigma(\mathbf{W}_i[\mathbf{h}(t-1), \mathbf{x}(t)] + \mathbf{b}_i) \tag{13b}$$

$$\tilde{\mathbf{c}}(t) = \tanh(\mathbf{W}_c[\mathbf{h}(t-1), \mathbf{x}(t)] + \mathbf{b}_c) \tag{13c}$$

$$\mathbf{c}(t) = \mathbf{f}(t) \otimes \mathbf{c}(t-1) + \mathbf{i}(t) \otimes \tilde{\mathbf{c}}(t) \tag{13d}$$

$$\mathbf{o}(t) = \sigma(\mathbf{W}_o[\mathbf{h}(t-1), \mathbf{x}(t)] + \mathbf{b}_o) \tag{13e}$$

$$\mathbf{y}(t) = \mathbf{h}(t) = \mathbf{o}(t) \otimes \tanh(\mathbf{c}(t)) \tag{13f}$$

À semelhança das redes recorrentes convencionais, o treinamento de uma LSTM também é realizado através de algoritmos de otimização baseados em derivadas da função custo propagadas ao longo da estrutura e ao longo do tempo, utilizando o algoritmo BPTT.

Resumidamente, as LSTMs manipulam o vetor de longo prazo $\mathbf{c}(t)$, aprendendo durante o treinamento o que deve ser guardado nele, o que deve ser descartado e o que deve ser aproveitado para gerar a saída $\mathbf{y}(t)$ e o vetor de curto prazo $\mathbf{h}(t)$. Dessa forma, podemos dizer que a atualização do vetor de estados $\mathbf{c}(t)$ é feita com o descarte de informações e a incorporação de novidades vindas da entrada.

Depois de uma exposição sobre os modelos preditores que utilizaremos nessa pesquisa, falaremos a seguir sobre dinâmica caótica, explicando qual a grande dificuldade de prever-se valores de uma série temporal gerada por um sistema desse tipo.

3 Sistemas com Dinâmica Caótica

Como dito anteriormente, sistemas com dinâmica cáotica se destacam pois, apesar de serem determinísticos, apresentam dependência sensitiva em relação às condições iniciais (DSCI). Dessa forma, duas trajetórias que partem de posições relativamente próximas no espaço de estados podem evoluir de uma forma totalmente distinta devido às não-linearidades presentes que amplificam as diferenças entre essas condições iniciais [4].

De forma resumida, a dinâmica caótica é marcada pela presença dos seguintes aspectos [7]:

1. Forte sensibilidade com respeito às condições iniciais;
2. A evolução temporal das variáveis de estado (parâmetros de ordem do sistema) é rápida e tem uma aparência errática;
3. Um sinal originado por um sistema caótico tem espectro de potências contínuo e de faixa larga;
4. Há uma produção de informação por parte do sistema;
5. Dão origem a atratores estranhos (estruturas topológicas que ditam a evolução temporal do fluxo de um sistema caótico) [17].

4 Próximos Passos

Como nessa primeira parte da iniciação o foco foi uma pesquisa bibliográfica dos temas a serem estudados nela, a segunda metade será voltada para a aplicação em si da predição das séries temporais de sistemas caóticos.

Para os experimentos computacionais, optamos pelas séries temporais do Mapa Logístico, descrito pelo cientista Robert May [18], o Mapa de Hénon, apresentado pelo astronômo e matemático francês Michel Hénon [19], a famosa série Mackey-Glass, dos cientistas Michael Mackey e Leon Glass [20], e o clássico Sistema de Lorenz, um dos mais incríveis e fundamentais trabalhos de sistemas caóticos, introduzido pelo matemático e meteorologista Edward Norton Lorenz [21], de forma a criar cenários diversificados para a análise do comportamento das redes neurais, com sistemas a tempo contínuo (Sistema de Lorenz e Equações de Mackey-Glass) e sistemas a tempo discreto (Mapa de Hénon e Mapa Logístico).

Em seguida, determinaremos aspectos mais fundamentais das redes neurais que serão utilizadas, como, por exemplo, a arquitetura empregada, assim como as métricas para o treinamento e análise. Para essa etapa, também planejamos um estudo das redes GRU (do inglês *Gated Recurrent Unit*) [22] e ESN (do inglês *Echo State Network*) [3], sendo que a última, em outros trabalhos de pesquisa, já indicou um bom desempenho preliminar no contexto de predição de séries temporais originadas por sistemas caóticos [12].

Após isso, faremos a aplicação das redes neurais à predição das séries escolhidas, avaliando a sensibilidade paramétrica de cada estrutura na busca das melhores configurações, a fim de traçar um quadro comparativo entre as técnicas consideradas.

Por fim, compilaremos os resultados no relatório final, de forma a conter uma discussão ampla e representativa dos ensaios realizados e das conclusões obtidas.

Referências

[1] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[2] J. T. Connor, R. D. Martin, and L. E. Atlas, "Recurrent neural networks and robust time series prediction," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 240–254, 1994.

[3] H. Jaeger, "Echo state network," *scholarpedia*, vol. 2, no. 9, p. 2330, 2007.

[4] N. Fiedler-Ferrara and C. P. C. do Prado, *Caos: uma introdução*. Editora Blucher, 1994.

[5] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[6] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019.

[7] R. R. de Faissol Attux, "Sobre dinâmica caótica e convergência em algoritmos de equalização autodidata," dissertação (mestrado), Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação, Campinas, SP, 2001.

[8] S. S. Haykin, *Adaptive filter theory*. Pearson Education India, 2008.

[9] S. Haykin, *Neural networks and learning machines, 3/E*. Pearson Education India, 2010.

[10] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[11] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[12] L. Boccato *et al.*, *Novas propostas e aplicações de redes neurais com estados de eco*. Tese (doutorado), Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação, Campinas, SP, 2013.

[13] T. Dozat, "Incorporating nesterov momentum into adam," 2016.

[14] Y. E. NESTEROV, "A method for solving the convex programming problem with convergence rate $O\left(1/k^2\right)$," *Dokl. Akad. Nauk SSSR*, vol. 269, pp. 543–547, 1983.

[15] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[16] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 10, pp. 2222–2232, 2016.

[17] D. Ruelle and F. Takens, "On the nature of turbulence," *Les rencontres physiciens-mathématiciens de Strasbourg-RCP25*, vol. 12, pp. 1–44, 1971.

[18] R. M. May, "Simple mathematical models with very complicated dynamics," *Nature*, vol. 261, pp. 459–467, jun 1976.

[19] M. Hénon, "A two-dimensional mapping with a strange attractor," *Communications in Mathematical Physics*, vol. 50, pp. 69–77, feb 1976.

[20] M. C. Mackey and L. Glass, "Oscillation and chaos in physiological control systems," *Science*, vol. 197, no. 4300, pp. 287–289, 1977.

[21] E. N. Lorenz, "Deterministic nonperiodic flow," *Journal of atmospheric sciences*, vol. 20, no. 2, pp. 130–141, 1963.

[22] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.